LVA 185.A03 Funktionale Programmierung (WS 20)

Leit- und Kontrollfragen III

Di, 27.10.2020

Stoff: Vorlesungsteil III – Kapitel 7, 8 und 9

Applikative Programmierung – Rekursion, Auswertung einfacher Ausdrücke, Programmentwicklung/Programmverstehen

(Ohne Abgabe, ohne Beurteilung; zur Selbsteinschätzung)

Teil III, Kapitel 7 'Rekursion'

1. Warum ist Rekursion für funktionale Sprachen (so viel) wichtiger als z.B. für objektorientierte Sprachen?
2. Worum geht es im Problem der *Türme von Hanoi*?
3. Vergleichen Sie die rekursive Implementierung der *Türme von Hanoi* mit einer iterativen (auf Schleifen basierenden) Lösung in einer imperativen oder objektorientierten Sprache. Versuchen Sie eine solche iterative Lösung erst selbst zu programmieren, bevor Sie nach einer veröffentlichten Lösung suchen. Welche Lösung scheint Ihnen konzeptuell und implementierungstechnisch einfacher? Die rekursive? Die iterative? Begründen Sie Ihre Antwort.
4. Welche Rekursionsmuster unterscheiden wir auf
   (a) mikroskopischer
   (b) makroskopischer

   Ebene?
5. Welchen Rekursionstyp haben die Implementierungen von:
   (a) `quickSort` (Kap. 7.1.1)
   (b) `hanoi` (Kap. 7.1.2)
   (c) `binom`, `binom'`, `binom''` (Kap. 1.1.1)
   (d) `zip` (Kap. 1.1.1)
   (e) `ggt` (Kap. 1.1.1)
   (f) `mod` (Kap. 1.1.1)
6. Was ist *Rechnen auf Parameterposition*? Wozu dient es?
7. Was ist ein *Aufrufgraph*? Wozu sind Aufrufgraphen nützlich?
8. Wie sind Aufrufgraphen aufgebaut, woraus bestehen sie?
9. Geben Sie den Aufrufgraph des folgenden Haskell-Skripts an:

```
fibSchritt :: (Integer,Integer) -> (Integer,Integer)
fibSchritt (m,n) = (n,m+n)

fibPaar :: Integer -> (Integer,Integer)
fibPaar n =
 | n == 0    = (0,1)
 | otherwise = fibSchritt (fibPaar (n-1))

fib' :: Integer -> Integer
fib' n = fst (fibPaar n)

fst :: (a,b) -> a
fst (x,y) = x
```

10. Was kann man am Aufrufgraphen aus der vorigen Teilaufgabe ablesen?
11. Gibt es weitere Eigenschaften, die man aus Aufrufgraphen ablesen kann, aber nicht anhand des obigen Beispiels? Welche?
12. Geben Sie ein Beipiel an, in dem sich diese Eigenschaften im Aufrufgraphen ablesen und demonstrieren lassen.
13. Was sind obere Schranken einer Funktion? Was untere?
14. Sind obere und untere Schranken einer Funktion eindeutig? Begründen Sie Ihre Antwort.
15. Wie kann man sich obere und untere Schranken von Funktionen graphisch veranschaulichen?
16. Was bedeutet es, dass eine Funktion f
    (a) höchstens
    (b) mindestens
    (c) genau

    von der Größenordnung einer Funktion g ist?
17. Wie, mit welchen Symbolen drückt man diese Beziehungen zweier Funktionen f und g aus?
18. Wie heißen diese Symbole? Nach wem sind sie benannt?
19. Was ist eine Komplexitätsklasse?
20. Geben Sie einige Beispiele wichtiger Komplexitätsklassen an.
21. Wie verändert sich für Funktionen dieser Komplexitätsklassen der Rechenaufwand, wenn sich die Eingabe um den Faktor
    (a) 10
    (b) 1.000
    (c) 1.000.000

    vergrößert?
22. Sie haben für ein Problem ein exponentielles Lösungsverfahren. Empfiehlt es sich, den schnellsten am Markt erhältlichen Rechner zu kaufen? Begründen Sie Ihre Antwort.
23. Der Berechnungsaufwand logarithmischer Verfahren wächst im Verhältnis zur Problemgröße äußerst langsam, der expnentieller Verfahren äußerst schnell. Können Sie diese Aussabe anhand von Zahlen aus Problemgröße und Operations-/Verfahrensschrittzahl untermauern und demonstrieren?
24. Nennen Sie je ein Beispiel für eine Funktion mit
    (a) konstanter
    (b) linearer
    (c) quadratischer
    (d) exponentieller

    Komplexität.
25. Warum betrachtet man asymptotische Schranken als Maß für die Berechnungskomplexität von Funktionen anstatt deren konkrete und deshalb exakten Aufwandsfunktionen?

Teil III, Kapitel 8 'Auswertung einfacher Ausdrücke'

1. Was ist mit
   (a) Expandieren
   (b) Simplifizieren

   von Ausdrücken gemeint?
2. Geben Sie je ein Beispiel für einen
   (a) Expansions-
   (b) Simplifikations-

   schritt eines Ausdrucks an.
3. Was ist ein Funktionsterm?
4. Welche Freiheitsgrade gibt es bei der Auswertung von Funktionstermen?
5. Illustrieren Sie diese(n) Freiheitsgrad(e) an (je) einem Beispiel.
6. Was legen der
   (a) applikative
   (b) normale

   Auswertungsstil für die Auswertung von Funktionstermen fest?
7. Vervollständigen Sie die
   (a) applikative
   (b) normale

   Auswertung des Funktionsterms `natSum 3` aus Beispiel 5 in Kapitel 8.2
8. Gegeben sind die musterbasierten Definitionen der Funktionen `fun91` und `fun91′`:

```
fun91 :: Integer -> Integer
fun91 n
 | n >  100 = n - 10
 | n <= 100 = fun91 (fun91 (n+11))

fun91':: Integer -> Integer
fun91' n =
 if n > 100 then n - 10 else fun91' (fun91' (n+11))
```

   Werten Sie die vier Funktionsterme
   (a) `fun91 101`
   (b) `fun91 100`
   (c) `fun91′ 101`
   (d) `fun91′ 100`

   für je einige wenige (!) Expansionsschritte
   (i) applikativ
   (ii) nomal

   aus.
9. Werte den Funktionsterm `fac 2`

(i) applikativ

(ii) nomal

für die musterbasierte Defintion der Fakultätsfunktion aus:

```
fac :: Integer -> Integer
fac 0 = 1
fac n = n * fac (n-1)
```

10. Führen Sie auch je einige Berechnungsschritte durch, die zur Lösung von

    (a) Übungsaufgabe 8.2.2

    (b) Übungsaufgabe 8.2.3

    (c) Übungsaufgabe 8.2.6

    aus Kapitel 8.2 nötig sind.

11. Was ist eine maximale Berechnungsfolge?

12. Sind maximale Berechnungsfolgen stets endlich? Begründen Sie Ihre Antwort.

13. Was sagt das nach Alonzo Church und John Barkley Rosser benannte Theorem über terminierende maximale Berechnungsfolgen aus?

14. Warum ist das Theorem von Church und Rosser für die Implementierung funktionaler Programmiersprachen wichtig?

15. Warum ist das Theorem von Church und Rosser auch für die Implementierung funktionaler (und auch nichtfunktionaler!) Programmiersprachen wichtig?

Teil III, Kapitel 9 'Programmentwicklung, Programmverstehen'

1. Lässt sich die Suche nach Algorithmen zur Lösung von Problemen automatisieren? Teilweise? Vollständig? Begründen Sie Ihre Antwort.

2. Welches Vorgehen schlägt Graham Hutton bei der Entwicklung von Programmen vor? Erläutern Sie knapp, aber präzise die verschiedenen Phasen seines Vorgehensvorschlags.

3. Wenden Sie das Verfahren von Graham Hutton an auf:

    (a) Die Berechnung der Länge einer Liste ganzer Zahlen.

    (b) Die Umkehrung einer Zeichenreihe.

    (c) Die Berechnung der Tiefe eines Binärbaums mit je einer ganzen Zahl als Benennung von Blättern und Verzweigungsknoten.

4. Vergleiche den Entwicklungsvorschlag von Graham Hutton mit dem von Norman Ramsey. Welche

    (a) Gemeinsamkeiten

    (b) Unterschiede

    gibt es? Ergänzen sich die Vorschläge? Sind sie konsistent miteinander oder widersprechen sie sich? Begründen Sie Ihre Antwort.

5. Gehen Sie die Probleme aus Aufgabe 3 nach dem Vorgehensvorschlag von Norman Ramsey an.

6. Welche Vorgehensweisen helfen beim Verstehen eines unbekannten Programmtexts?

7. Gegeben ist die Deklaration:

```
mapWhile :: (a -> b) -> (a -> Bool) -> [a] -> [b]
mapWhile f p [] = []
mapWhile f p (x:xs)
 | p x       = f x : mapWhile f p xs
 | otherwise = []
```

Was ist die Bedeutung von `mapWhile`? Erklären Sie knapp, aber gut nachvollziehbar, was sich aus den einzelnen Zeilen der Deklaration von `mapWhile` herauslesen lässt und wie die Auswertung der Funktion vorgeht.

8. Das erste und zweite Argument von `mapWhile` sind funktional, sind Funktionen. Für das zweite Argument gibt es noch eine genauere Bestimmungsbezeichnung als Funktion. Welchen? Gibt es noch weitere Bezeichnungen für diesen Begriff?

9. Gegeben ist die Deklaration:

```
filterMap :: [a] -> (a -> Bool) -> (a -> b) -> [b]
filterMap [] _ _ = []
filterMap (x:xs) p f
 | p x  = (f x) : filterMap xs p f
 | True = filterMap xs p f
```

Was ist die Bedeutung von `filterMap`? Erklären Sie knapp, aber gut nachvollziehbar, wie die Auswertung der Funktion vorgeht.

10. Welche Laufzeitkomplexität (linear, quadratisch, logarithmisch,... in der Zahl, in der Größe von...) haben

    (a) `mapWhile`?

    (b) `filterMap`?

Teil I – II, Verschiedene Kapitel

1. Wodurch unterscheiden sich die Typen `Int` und `Integer` voneinander?

2. Welche vordefinierten Operatoren gibt es auf Wahrheitswerten? Nennen Sie einige Beispiele.

3. Tupelwerte sind i.a. heterogen, Listenwerte homogen. Was bedeutet das?

4. Welche Liste repräsentiert der Ausdruck `[1,3..10]`?

5. Welches ist die Grunddarstellung des 'syntaktisch gezuckerten' Ausdrucks `[1..4]`?

6. Welchen Wert haben folgende Ausdrücke?

    (a) `map (+1) [-2..2]`

    (b) `map (1+) [-2..2]`

    (c) `map inc [-2..2] where inc n = n+1`

    (d) `map inc [-2..2] where inc = \n -> n+1`

    (e) `map inc [-2..2] where inc = (+1)`

7. Welche der folgenden Ausdrücke sind gültig, welche nicht?

    (a) `["sin","cos","tan"]`

    (b) `["sin",cos,"tan"]`

    (c) `[sin,cos,tan]`

    (d) `["sin",[],"tan"]`

    (e) `["sin","","tan"]`

    (f) `["sin"," ",""tan"]`

(g) `["sin",‘ ’,"tan"]`

(h) `[‘sin’,‘ ’,‘tan’]`

Geben Sie für gültige Ausdrücke den Typ an; für nichtgültige Ausdrücke eine knappe Begründung, warum sie nicht gültig sind.

8. Was sind Listenaufzählungsausdrücke? Illustrieren Sie Ihre Antwort mit einigen Beispielen.

9. Wie sehen die Muster(ausdrücke) aus, die Listen genau der folgenden Werte beschreiben, die ‘darauf passen’?

   (a) Listen mit genau

      i. keinem
      ii. einem
      iii. zwei
      iv. drei

   Element(en)?

   (b) Listen mit mindestens

      i. einem
      ii. zwei
      iii. drei

   Element(en)?

10. Gibt es Muster(ausdrücke), auf die genau Listen folgender Werte passen, die Listen genau dieser Werte beschreiben?

    (a) Listen mit höchstens

       i. einem
       ii. zwei
       iii. drei

    Element(en).

    (b) Listen mit einer beliebigen Zahl von Elementen

    Geben Sie den entsprechenden Musterausdruck an oder erläutern Sie, warum es keinen entsprechenden Musterausdruck gibt.

11. Wozu dienen `let`- und `where`-Deklarationen? Wodurch unterscheiden Sie sich? Geben Sie je ein illustrierendes Beispiel an.

12. Zeigen Sie, dass die linke und rechte Seite der definierenden Gleichung von `curry` denselben Typ besitzen:

```
curry :: ((a,b) -> c) -> a -> b -> c
curry f x y = f (x,y)
```

13. Ergänzen Sie Klammern in der obigen Definition von `curry` so, dass Signatur und definierende Gleichung jeweils vollsändig, aber nicht überflüssig geklammert sind.

14. Wie sind `newtype`-Deklarationen aufgebaut? Erklären Sie die einzelnen Bestandteile einer `newtype`-Deklaration und wie sie heißen?

15. Algebraische Datentypen in Haskell erlauben neue numerische Typen einzuführen, z.B. für komplexe Zahlen. Um andere, nicht vordefinierte Datentypen einzuführen, braucht man Haskells Konzept der Neuen Typen. Richtig oder falsch. Begründen Sie Ihre Antwort.

16. Wann immer eine `type`-Vereinbarung möglich ist, ist auch eine `data`-Vereinbarung möglich

    (a) und umgekehrt.

(b) aber nicht umgekehrt.

Richtig, falsch, sinnvoll gefragt, nicht sinnvoll gefragt? Begründen Sie Ihre Antwort.

17. Welcher Frage ist John W. Backus in seiner Preisrede zur Verleihung des *Turing Awards 1977* nachgegangen?

18. Was meint *Rechnen mit Funktionen*?

19. Was sind Beispiele aus der Mathematik (auch der Schulmathematik) für das *Rechnen mit Funktionen*?

20. Der *Algorithmus von Euklid*, das *Sieb des Eratosthenes*. Was ist älter?

21. Was sind Protoimplementierungen einer Typklasse? Wozu sind sie nützlich?

22. Wofür gibt es eine Protoimplementierung in der Typklasse `Eq`?

23. Grunsätzlich kann jeder Typ in Haskell zur Instanz einer Typklasse gemacht werden, gleich ob mit `type`, `newtype` oder `data` eingeführt. Richtig oder falsch? Falls falsch, was genau?

24. Machen Sie den Typ:

```
data NaturalNumber = Null | Succ NaturalNumber
```

unter bestmöglicher Ausnutzung von Protoimplementierungen zu einer Instanz der Typklassen:

(a) `Eq`
(b) `Show`
(c) `Ord`
(d) `Num`
(e) `Enum`

Überlegen Sie sich, wie Sie möglichst sinnvoll bei der Implementierung von Funktionen vorgehen, die auf den ganzen statt natürlichen Zahlen negative Ergebnisse liefern würden. Die Funktion `show` soll in üblicher Weise als Folge von Dezimalziffern ausgeben.

25. Wiederholen Sie die vorige Aufgabe für den Typ:

```
data IntegerNumber = Null | Succ IntegerNumber | Pred IntegerNumber
```

Wie gehen Sie mit der Nichteindeutigkeit der Darstellung ganzer Zahlen als `IntegerNumber`-Werte bei den Instanzbildungen um? Benutzen Sie eine 'Normaldarstellung' für `IntegerNumber`-Werte für Funktionsergebnisse? Wie sieht Ihre Normaldarstellung aus? Wie ist sie definiert?

26. Wenn Sie eine Normaldarstellung benutzen, schreiben Sie eine Wahrheitsfunktion:

```
is_nf :: IntegerNumber -> Bool
```

die überprüft, ob ein `IntegerNumber`-Wert in Normalform ist oder nicht.

27. Modifizieren Sie Ihre Instanzbildungen aus der vorvorausgehenden Aufgabe so, dass Funktionen angewendet auf Werte in Normalform wieder Werte in Normalform liefern, ansonsten mit einer Fehlermeldung abbrechen.

28. Wenn Sie eine Normaldarstellung benutzen, schreiben Sie eine Transfomrationsfunktion:

```
to_nf :: IntegerNumber -> IntegerNumber
```

die einen `IntegerNumber`-Wert in seine Normalform überführt.

29. Ändern Sie die ursprünglichen Instanzbildungen für `IntegerNumber` so ab, dass Funktionen ihre Argumente in Normalform überführen und ihre Ergebnisse ebenfalls in Normalform liefern.

30. Was ist Haskells Feldsyntax? Wozu ist sie nützlich?